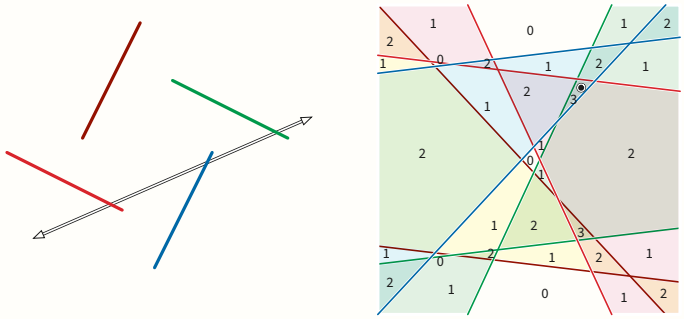
1. Suppose you are given a set S of n line segments in the plane. Describe and analyze an algorithm to find a line that intersects as many segments in S as possible. *[Hint: What is the dual of a line segment?]*

Solution: We solve a dual formulation of the problem, using the standard duality $(a, b) \leftrightharpoons y = ax - b$. A line segment $s = \overline{pq}$ is the set of all points between one endpoint p and the other endpoint q on the line $\overleftrightarrow{pq}$. Thus, the dual s^* of s is the set of all lines between one line p^* and another line q^* that pass through the dual point $(\overleftrightarrow{pq})^*$. This set of lines sweeps out a double wedge through $(\overleftrightarrow{pq})^*$, with p^* and q^* on its boundary. Another line ℓ intersects s if and only if the dual point ℓ^* lies inside the double wedge s^*; that is, either ℓ^* above p^* and below q^*, or ℓ^* is above q^* and below p^*. So let's put on our duality glasses: **Given a set S^* of n double-wedges, we want to find a point that lies in as many double-wedges in S^* as possible.**



Let L be the set of $2n$ bounding lines of these wedges (dual to the endpoints of segments in S). We can construct the arrangement of L in $O(n^2)$ time. Each edge e of the arrangement lies on the boundary of exactly one double-sedge in S^*. We direct every edge e in the arrangement as follows: If e lies on the upper boundary of its double wedge, direct e from right to left; otherwise, direct e from left to right. (We can distinguish these two cases in $O(1)$ time.)

For each face f in the arrangement, let $W(f)$ denote the number of double wedges that contain f. When f is the face bounded by the upper envelope, we have $W(f) = 0$. Let $\mathsf{above}(e)$ and $\mathsf{below}(e)$ respectively denote the faces just above and below any edge e in the line arrangement. If edge e is directed from right to left, we have $W(\mathsf{below}(e)) = W(\mathsf{above}(e)) + 1$; otherwise, we have $W(\mathsf{below}(e)) = W(\mathsf{above}(e)) - 1$. Thus, we can compute $W(f)$ for every face f in $O(n^2)$ time, by traversing the dual graph of the line arrangement. (Essentially we are treating the $2n$ wedges as curves and computing winding numbers.)

Finally, we return the line dual to any point in any face f that maximizes $W(f)$. The entire algorithm runs in $\boldsymbol{O(n^2)}$ ***time***. ■

Rubric: 10 points = 4 for dual problem formulation + 4 for Alexander numbering dual arrangement + 2 for running time

2. Let L be a set of n lines in the plane in general position.

   (a) Prove that $\sum_f \deg(f)^2 = O(n^2)$, where the sum is over all faces f in the arrangement of L, and $\deg(f)$ denotes the number of edges of face f.

Solution: This bound is a direct consequence of the Zone Theorem. Let ∂f denote the set of edges on the boundary of any face f; then $\deg(f) = |\partial f|$. For any line ℓ, let $\mathsf{zone}(\ell)$ denote the zone of ℓ in the arrangement of L.

$$\begin{aligned}
\sum_f \deg(f)^2 &= \sum_f \sum_{e \in \partial f} \deg(f) && \text{[definition of } \deg(f)\text{]} \\
&= \sum_e \sum_{f : e \in \partial f} \deg(f) && \text{[addition is commutative]} \\
&= \sum_{\ell \in L} \sum_{e \subseteq \ell} \sum_{f : e \in \partial f} \deg(f) && \text{[clustering edges along lines]} \\
&= \sum_{\ell \in L} \sum_{f \in \mathsf{zone}(\ell)} \deg(f) && \text{[definition of zone]} \\
&= \sum_{\ell \in L} O(n) && \text{[zone theorem]} \\
&= O(n^2)
\end{aligned}$$

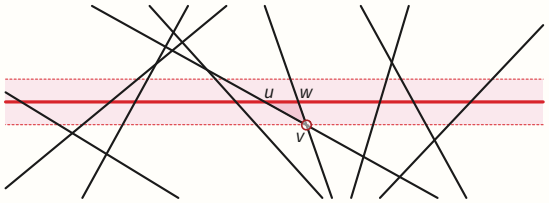
■

Rubric: 5 points.

(b) Prove that the arrangement of L contains $\Omega(n)$ bounded triangular faces.

Solution: I claim that if $\ell \geq 3$, then each line in L is incident to at least one bounded triangular face. This claim immediately implies that the arrangement of L contains at least $n/3$ triangular faces (unless $n \leq 2$, but whatever).

Fix an arbitrary line $\ell \in L$. The arrangement of L contains at least one vertex that is not on ℓ. Let v be the arrangement vertex that has minimum non-zero distance to ℓ. Finally, let u and w be the intersection points of ℓ and the two lines that intersect at v. To complete the proof, it suffices to show that the triangle uvw is a face of the arrangement.



Let ℓ' be any other line in L. If ℓ' intersects segment vw, it must intersect at least one other edge of the triangle uvw. If ℓ' intersects uv, the arrangement vertex $uv \cap \ell'$ is closer to ℓ than v is, contradicting the definition of v. Symmetrically, if ℓ' intersects vw, the intersection point is a closer arrangement vertex. We conclude that ℓ' does not intersect uvw. It follows that uvw is a face of the line arrangement, which completes the proof.

A stronger version of this theorem states that the *projective* arrangement of any $n \geq 3$ lines in general position contains at least n triangular faces. In a projective arrangement, edges and faces "wrap around" at infinity; for example, the upper and lower envelopes are boundaries of the same projective face. The stronger theorem is proved by fixing an arbitrary line ℓ, applying a projective transformation to move ℓ to infinity, and then arguing that any vertex of the convex hull of the arrangement vertices not on ℓ defines a triangular face with ℓ. Assuming $n \geq 4$, this convex hull has at least three vertices, so ℓ is incident to at least three triangular faces. (When $n = 3$, all four faces of the projective arrangement are triangular. ■

Rubric: 5 points.

3. Let P be a set of *moving* points in the plane, each represented by a starting position and a fixed velocity vector. For any real number t, a point with starting position (a, b) and velocity (u, v) is located at $(a + tu, b + tv)$ at time t. As the points in P move through the plane, their axis-aligned bounding box continuously changes.

   (a) Describe an algorithm to compute the time t when the bounding box of the moving points has smallest *perimeter*.

Solution: Suppose each moving point is specified as a quadruple (a_i, b_i, u_i, v_i), where (a_i, b_i) is the position of the ith point at time 0 and (u_i, v_i) is the ith point's fixed velocity.

The solution vector $(t, x_{\max}, x_{\min}, y_{\max}, y_{\min})$ for the following linear program describes the time t and the coordinates of the optimal bounding box.

$$\begin{array}{lll} \text{minimize} & x_{\max} - x_{\min} + y_{\max} - y_{\min} & \\ \text{subject to} & a_i + tu_i \le x_{\max} & \text{for all } i \\ & a_i + tu_i \ge x_{\min} & \text{for all } i \\ & b_i + tv_i \le y_{\max} & \text{for all } i \\ & b_i + tv_i \ge y_{\min} & \text{for all } i \end{array}$$

We can solve this five-variable linear program in ***O(n) expected time*** using Seidel's algorithm. ■

Rubric: 5 points.

   (b) Describe an algorithm to compute the time t when the bounding box of the moving points has smallest *area*.

Solution: Index the points in P as $p_1, p_2, \ldots, p_n$, and suppose each point p_i is represented by a quadruple (a_i, b_i, u_i, v_i), where (a_i, b_i) is the location of p_i at time 0, and (u_i, v_i) is p_i's fixed velocity..

At all times, the bounding box of P is determined by four points, one on each edge of the box. (These points are not necessarily distinct; a point in the northwest corner of the box lies on both the north edge and the west edge, for example.) At a high level, our algorithm considers all quadruples of points that ever determine the bounding box, and for each quadruple, computes the minimum-area bounding box while those four points are extreme.

Let's start by considering the point on the north edge of the box, that is, the point with maximum y-coordinate. The y-coordinate of p_i is defined by the function $y = b_i + t \cdot v_i$; the graph of this function in the (t, y)-plane is a straight line. By definition, the upper envelope of these n lines is graph of the function $y = \max_i(b_i + t \cdot v_i)$. The upper envelope is an unbounded convex polygon with at most n edges, and therefor at most $n - 1$ vertices. Each edge of the upper envelope describes an interval of time where a particular point has maximum y-coordinate. Each vertex of the upper envelope describes an *event* where this uppermost point changes. We can compute this upper envelope in $O(n \log n)$

time, using (the dual of) any planar convex hull algorithm.

Symmetric arguments imply that there are at most $n-1$ events where the lowest point changes, at most $n-1$ events where the leftmost point changes, and at most $n-1$ events where the rightmost point changes. Thus, altogether, there are at most $4n-4$ events where the points determining the bounding box change; these events divide the time line into at most $4n-3$ intervals.[a] We can compute the sorted sequence of events, as well as the four extreme points for each time interval, in $O(n \log n)$ time.

Within each time interval, the area of the bounding box is a quadratic function of time:

$$\begin{aligned}
Area(t) &= \big((a_E + t\,u_E) - (a_W + t\,u_W)\big) \cdot \big((b_N + t\,v_N) - (b_S + t\,v_S)\big) \\
&= \big((a_E - a_W) + t(u_E - u_W)\big) \cdot \big((b_N - b_S) + t(v_N - v_S)\big) \\
&= t^2(u_E - u_W)(v_N - v_S) \\
&\quad + t\big((a_E - a_W)(v_N - v_S) + (u_E - u_W)(b_N - b_S)\big) \\
&\quad + (a_E - a_W)(b_N - b_S)
\end{aligned}$$

(Here, for example, (a_E, b_E) and (u_E, v_E) are the starting location of the point on the East edge of the bounding box.) This function is minimized either at one of the endpoints of the time interval, or at the critical moment where its derivative is zero (if that moment lies within the interval):][b]

$$\frac{d}{dt}Area(t) = 0 \iff t = \frac{(a_E - a_W)(v_N - v_S) + (u_E - u_W)(b_N - b_S)}{-2(u_E - u_W)(v_N - v_S)}$$

Thus, we can compute the minimum bounding-box area during any interval in $O(1)$ time.

The overall algorithm runs in $\boldsymbol{O(n \log n)}$ ***time***. ■

[a] More careful analysis implies that the number of events is at most $2n$. But this observation only reduces the running time by a constant factor.

[b] More careful analysis implies that in fact, any critical point in the interior of a time interval must be a local *maximum* for area, so it suffices to check only the event times. But this optimization only reduces the running time by a constant factor.

Rubric: 5 points.

4. In class we saw the classical *funnel* algorithm to compute shortest paths inside a triangulated simple polygons. How would you modify this algorithm to find shortest paths in a polygon with holes?

   (a) Describe and analyze an algorithm to compute the shortest path between two given points in the interior of a polygon with *one* hole. *[Hint: Which way does the path go around the hole?]*

   (b) Describe and analyze an algorithm to compute the shortest path between two given points in the interior of a polygon with *two* holes.

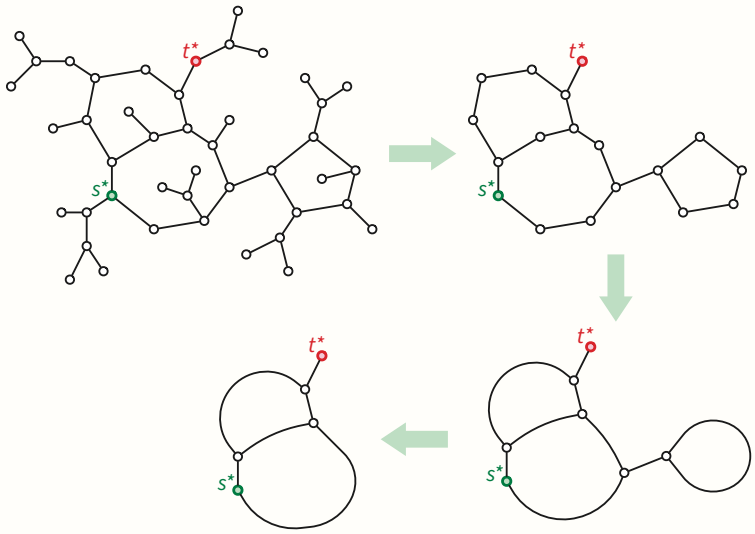
   (c) **[Extra credit]** Describe and analyze an algorithm to compute shortest paths in a polygon with h holes; analyze your algorithm as a function of both n (the total number of polygon vertices) and h (the number of holes).

   In all cases, you can assume that you are given a triangulation of the input polygon. For full credit, your algorithms should run in $O(n)$ time (for any constant h).

Solution: Suppose we are given a triangulation T of a polygon P with h holes, along with two points s and t in P. The shortest path from s to t crosses each diagonal in the triangulation at most once. Thus, the sequence of triangles that the shortest path intersects describes a simple path (meaning no repeated vertices or edges) in the dual graph T^*.

At a high level, our algorithm considers all simple paths in T^* with the correct endpoints; for each such path, we run the standard funnel algorithm to find the shortest path from s to t through the corresponding sleeve of triangles in T. The running time of the algorithm is $O(Nn)$, where N is the number of dual paths the algorithm considers. I will prove that $N = O(8^h)$.

To efficiently enumerate simple paths in T^*, we first *reduce* T^* as follows. Let s^* and t^* denote the nodes in T^* dual to the triangles containing s and t, respectively.



- First we repeatedly remove vertices of degree 1 *except* s^* *and* t^*, until no such vertices remain. This operation is sometimes called a *leaf reduction*.

- Then we repeatedly replace paths through degree-2 vertices *except* s^* *and* t^* with single edges, until no such paths remain. (his operation is sometimes called a *series reduction*. As we perform series reductions, we maintain the path in T^* corresponding to each edge the remaining graph.
- The final phase is optional: If the remaining graph contains bridge—an edge whose deletion disconnects the graph—that does *not* separate s and t, we delete the bridge and the component not containing s and t, and then (if possible) perform another series reduction at the remaining endpoint of the bridge.

The entire reduction process can be performed in $O(n)$ time. Every simple path from s^* to t^* in the reduced dual graph R corresponds to a simple path from s^* to t^* in T^* and vice versa.

The reduced dual graph R is a planar graph with at most $h+1$ faces—one bounded face for each hole in P, plus the unbounded outer face. Every vertex has degree 3 except possible s^* and t^*, which might have degree 1 or 2. Routine calculations with Euler's formula now imply that R has at most $2h+2$ vertices and at most $3h+1$ edges.

Every simple path in R uses a subset of the edges; it follows immediately that R contains at most $2^{3h+1} = O(8^h)$ simple paths from s^* to t^*. Moreover, we can enumerate these paths in $O(8^h h)$ time by brute force: For every subset S of edges of R, test whether S is a simple path from s^* to t^* in $O(h)$ time.

For each simple path in R, we recover the corresponding path in T^* and run the funnel algorithm in the corresponding sleeve in T in $O(n)$ time. Finally, we return the shortest of all the funnel paths. The overall algorithm runs in $\boldsymbol{O(8^h n)}$ ***time***, which is $O(n)$ time for any constant h.

We can do better, even without reduction. The symmetric difference between between any two paths from s^* to t^* in T^* is a subgraph of T^* in which every vertex has degree 2; this subgraph must be the union of disjoint simple cycles. Any union of disjoint cycles is the boundary of the union of a subset of the bounded faces of R, namely, the faces with odd depth/winding number. It follows that the number of simple paths from s^* to t^* is at most 2^h. (It should be easy to see that this bound is tight in the worst case.) We can enumerate these paths by brute force in $O(2^h n)$ time. For each subset of bounded faces, compute the boundary of its union, compute the symmetric difference of that boundary with some fixed path from s^* to t^*, and finally test whether that symmetric difference is connected, all in $O(n)$ time. (Using the reduced graph R would reduce the running time of this part of the algorithm from $O(2^h n)$ to $O(2^h h)$.) Finally, for each simple path from s^* to t^* in T^*, we can run the funnel algorithm in the corresponding sleeve in T in $O(n)$ time. The overall algorithm runs in $\boldsymbol{O(2^h n)}$ ***time***, which is $O(n)$ time for any constant h. ■

Rubric: 15 points = 5 for each part. A correct algorithm for part (b) implies part (a), and a correct algorithm for part (c) implies parts (a) and (b). Any running time of the form $f(h) \cdot O(n)$ for part (c) is worth full credit. This is not the fastest algorithm known for part (c); in particular, part (c) can be solved in $O(n \log n)$ time for any h!